开篇词 | C++这么难，为什么我们还要用C++？

2019-11-25 吴咏炜

现代C++实战30讲

进入课程 >

讲述：吴咏炜

时长 15:36　大小 10.72M

你好，我是吴咏炜。

C++ 可算是一种声名在外的编程语言了。这个名声有好有坏，从好的方面讲，C++ 性能非常好，哪个编程语言性能好的话，总忍不住要跟 C++ 来单挑一下；从坏的方面讲，它是臭名昭著的复杂、难学、难用。当然，这样一来，熟练的 C++ 程序员也就自然而然获得了“水平很高”的名声，所以这也不完全是件坏事。

不管说 C++ 是好还是坏，不可否认的是，C++ 仍然是一门非常流行且非常具有活力的语言。继沉寂了十多年，并终于发布语言标准的第二版——C++11——之后，C++ 以每三年一版的频度发布着新的语言标准，每一版都在基本保留向后兼容性的同时，提供着改进和新功能。**本专栏主要就是讲这些新特性以及相关的编程实践。**

在讲所有这些细节之前，我想先讲一讲为什么要用 C++，什么时候该用 C++，及如何学习 C++。

C++ 的意义

C++ 程序员应该都听到过下面这种说法：

> C++ 是一门多范式的通用编程语言。

多范式，是因为 C++ 支持面向过程编程，也支持面向对象编程，也支持泛型编程，新版本还可以说是支持了函数式编程。同时，上面这些不同的范式，都可以在同一项目中组合使用，这就大大增加了开发的灵活性。因此，C++ 适用的领域非常广泛，小到嵌入式，大到分布式服务器，到处可以见到 C++ 的身影。

下面是一些著名的用到 C++ 的场合：

- 大型桌面应用程序（如 Adobe Photoshop、Google Chrome 和 Microsoft Office）
- 大型网站后台（如 Google 的搜索引擎）
- 游戏（如 StarCraft）和游戏引擎（如 Unreal 和 Unity）
- 编译器（如 LLVM/Clang 和 GCC）
- 解释器（如 Java 虚拟机和 V8 JavaScript 引擎）
- 实时控制（如战斗机的飞行控制和火星车的自动驾驶系统）
- 视觉和智能引擎（如 OpenCV、TensorFlow）
- 数据库（如 Microsoft SQL Server、MySQL 和 MongoDB）

有些同学可能会觉得，这些应用场景似乎和平时的开发场景有点远啊！你的感觉是对的。有些传统上使用 C++ 的场合现在已经不一定使用 C++，最典型的是个人电脑上的桌面应用。以前 Windows 下开发桌面应用常常用 MFC，微软的 C++ 框架，而现在我估计听说过 MFC 的程序员都不多吧。目前很流行的 Visual Studio Code 主要是用 TypeScript 写的，不是 C++。而我自己也用 C# 写过桌面应用，不过界面逻辑之外的计算和处理仍然是用一个 C++ 的 DLL 来完成。典型情况是，需要性能的组件用 C++ 来写，整个应用程序融合多种不同的语言。

前面我提到了， C++ 的传统领域有被侵蚀的风险，那是因为和它相竞争的语言远远不止一个，可以说是上下夹攻。

如果专注性能和最小内存占用的话，C 仍然是首选——嵌入式领域用 C 非常多，而 Linux 也是用纯 C 写的。

如果专注抽象表达和可读性的话，那 Python 之类的脚本语言则要方便得多。

图形界面（GUI）编程传统上是 C++ 的地盘，但近年来 C# 和 JavaScript 占领了很大一部分市场。

游戏算是 C++ 的经典强项了，但有了 C++ 写的游戏引擎，游戏用 C# 写也没啥问题了——你可能不一定知道，Unity 游戏引擎上的首选开发语言是 C#，而王者荣耀是用什么游戏引擎呢？答案正是 Unity——所以王者荣耀可以认为是用 C# 开发的。

还有，Go 和 Rust 也加入了战团，对 C++ 形成了一定的竞争……

看起来，C++ 有点危险啊……

不过，真是这样吗？我们需要回到 C++ 的核心竞争力上来看一下。

抽象能力：意味着较高的开发效率，同时，更重要的是，不会因抽象而降低性能。

性能：这不用多说了，就是快并且占用资源少。

功耗：这是近年来我们越来越关注的问题，跟性能直接相关，性能好了功耗自然就低。

计算机在发明的初期，价格奇高，而性能拿今天的标准来看却是极低的，自然不能不关注性能。慢慢地，计算机的性能“足够”了，性能似乎也就不那么重要了，脚本语言于是也有了用武之地。而随着移动设备的普遍使用，大量设备用电池供电而不接电源了，功耗就逐渐成了我们大家关注的大问题。因此，即使主流移动平台的开发语言不是 C++——而是 Java 和 Objective-C 或 Swift——但任何性能要求高的应用，都几乎必然会用到 C++ 开发的组件。

同时，移动设备要联网，也大大刺激了服务器的增加。在服务器端，虽然没有电池电量的问题，但有着服务器集群的供电问题、空调问题、需要的服务器数量问题等，因而 C++ 的使用也是非常广泛的。

前面说到了王者荣耀的客户端是用 Unity + C# 开发的，但我没有说王者荣耀的服务器端——那可还是用 C++ 开发的。另外，有一点我前面还藏着呢！虽然王者荣耀初期是纯用 Unity 开发的，没有用到 C++；但后来，腾讯又用 C++ 把游戏的逻辑部分独立成了一个 GameCore，进一步提高了性能 [1]。

目前，跟 C++ 定位差不多、能有直接竞争关系的，也就是既支持高度抽象、又追求高性能的通用编程语言，其实只有 Rust 一种。而 Rust 远没有达到跟 C++ 一样的成熟和普及程度。这也可以从 TIOBE 的排名看出来：C++ 是第 4 位，而 Rust 是第 25 位 [2]。

另外，和 C 的兼容性，也是 C++ 的一大优势。虽然现在很多大型程序都混杂了多种语言，但在小项目里，减少语言的数量可以简化开发和部署。前不久，我在 Python 里做了一些加解密运算，发现使用的第三方库性能仍不够高，虽然它已经用了 C 开发的加解密引擎。所以，我找了用 C 写的高性能加解密代码，然后使用 pybind11 库 [3]，只手写了一百来行的 C++11 代码，就把性能又提高了几倍。

什么时候该用 C++?

如此说来，C++ 既然性能又好，又支持抽象，为什么没有更流行呢?

因为代价更高。C++ 是一种复杂的语言，难以上手和熟练掌握，因此也是一种比较容易出错、被误用的语言。C++ 一直与 C 基本保持了向后兼容性，这种兼容性，也一直是 C++ 的安全性和易用性方面的负担。C++ 比起 C 来，要更安全，更不容易出现缓冲区溢出这类漏洞，但跟没有指针概念的语言比起来，它仍然是一种“不安全”的语言。我的个人经验，完成同样的功能，C++ 需要的代码行数一般是 Python 的三倍左右，而性能则可以达到 Python 的十倍以上。

问题来了：你在开发上额外付出的时间，能从性能上省回来吗?

显然，这取决于你开发软件的用途和开发时间。举个例子，如果你用 Python 开发需要一天，运行需要十秒，并且不需要反复运行；那么，转用 C++ 开发就意味着开发费用也许要增加两倍，开发加运行的总时间增加两天，大亏。

反之，如果用 Python 开发还是需要一天，单次运行需要十秒，但是软件会作为服务长时间运行、每天被调用十万次。在这种情况下，明显你就需要多台服务器来支撑其使用了。这

时，如果用 C++ 开发会需要额外的两天，但跟 Python 相比，部署上有望节约十分之九的硬件和电费——那就很值了。

简言之，当你的软件属于运算密集或者内存密集型，你需要性能、且愿意为性能付出额外代价的时候，应该考虑用 C++，特别在你的代码需要部署在多台服务器或者移动设备的场合。反之，如果性能不会成为你开发的软件的瓶颈，那 C++ 可能就不是一个最合适的工具。

此外，在嵌入式应用的场景，那就根本不是值不值、而是行不行的问题。如果程序完成一个功能不能在指定的若干毫秒、甚至微秒内完成，那产品根本是失败、不可用的。在这种场合，能和 C++ 竞争的只有 C，但 C 是一种开发效率更低、更需要堆人力的语言了。在嵌入式开发使用 C++ 的最大障碍可能不是技术，而是人力资源——搞嵌入式开发的程序员可能大多都习惯使用纯 C 了。

由于 C++ 是解决性能问题的利器，短时间里在市场上没有真正的竞争对手，对 C++ 的需求会在相当长的时间里一直存在，尤其在大公司和像金融机构一样对性能渴求的地方。顺便提一句，C++ 之父 Bjarne Stroustrup 目前就职的地方便是摩根斯坦利。

如何学习 C++?

作为很多聪明人使用过的语言，C++ 在某些场合也可能被用来炫技，写出除了本人之外谁都看不懂的高抽象代码。这恰恰是 Bjarne 想努力抵制的方向。他想让 C++ 对初学者变得更为友好，也明确提出过，他不希望 C++ 是一种让人们耍机灵的语言，而是一种让人们更易于使用的语言 [4]。

这同样也是本专栏的一个目标：我希望你能把 C++ 当作一种实用的语言，能用它写出抽象但自然的代码，而非佶屈聱牙、难以卒读的那种。希望我 30 年的 C++ 经验能够给你一点帮助。

学习 C++ 语言就像学一门活跃使用中的外语，你不要期望能够掌握所有的单词和语法规则——那对于世界上 99.999999% 的人来说是不可能的。但语言是服务于人的，语法规则也是服务于人的，是为了让人们能够更好地沟通和表达。虽然 C++ 的每一个新标准都是让语言从定义和规则的角度变得更复杂，但从用法上来说，新标准允许人们能够更简单地表达自己的计算意图。跟学外语一样，我们需要的是多看多写，掌握合适的“语感”，而不是记住所有的规则。

Bjarne 有一个洋葱理论：抽象层次就像一个洋葱，是层层嵌套的。如果想用较低的抽象层次表达较高的概念，就好比一次切过了很多层洋葱，你会把自己的眼泪熏出来的。与这个思路相反，教 C++ 往往有一种不好的倾向，从那些琐碎易错的底层教起，自底向上，使得学生常常在尚未领悟到抽象的真谛之前就已经被 C++ 的复杂性吓翻，从入门到放弃；或者，在学了基本的 C 语法和 class 之后就满足了，错过了高级抽象带来的全新境界。他主张学习应当自顶向下，先学习高层的抽象，再层层剥茧、丝丝入扣地一步步进入下层。如果一次走太深的话，挫折可能就难免了。

作为专栏而非具体的工具参考书，我会重点讲是什么和为什么，而不是语法细节。同样，我也不讲或少讲技巧，但在需要的地方，我会给出合适的参考资料。希望你在学习了本专栏之后，能够知道某个 C++ 的功能为什么存在和应该在什么情况下使用。那样的话，本专栏的目的就达到了。

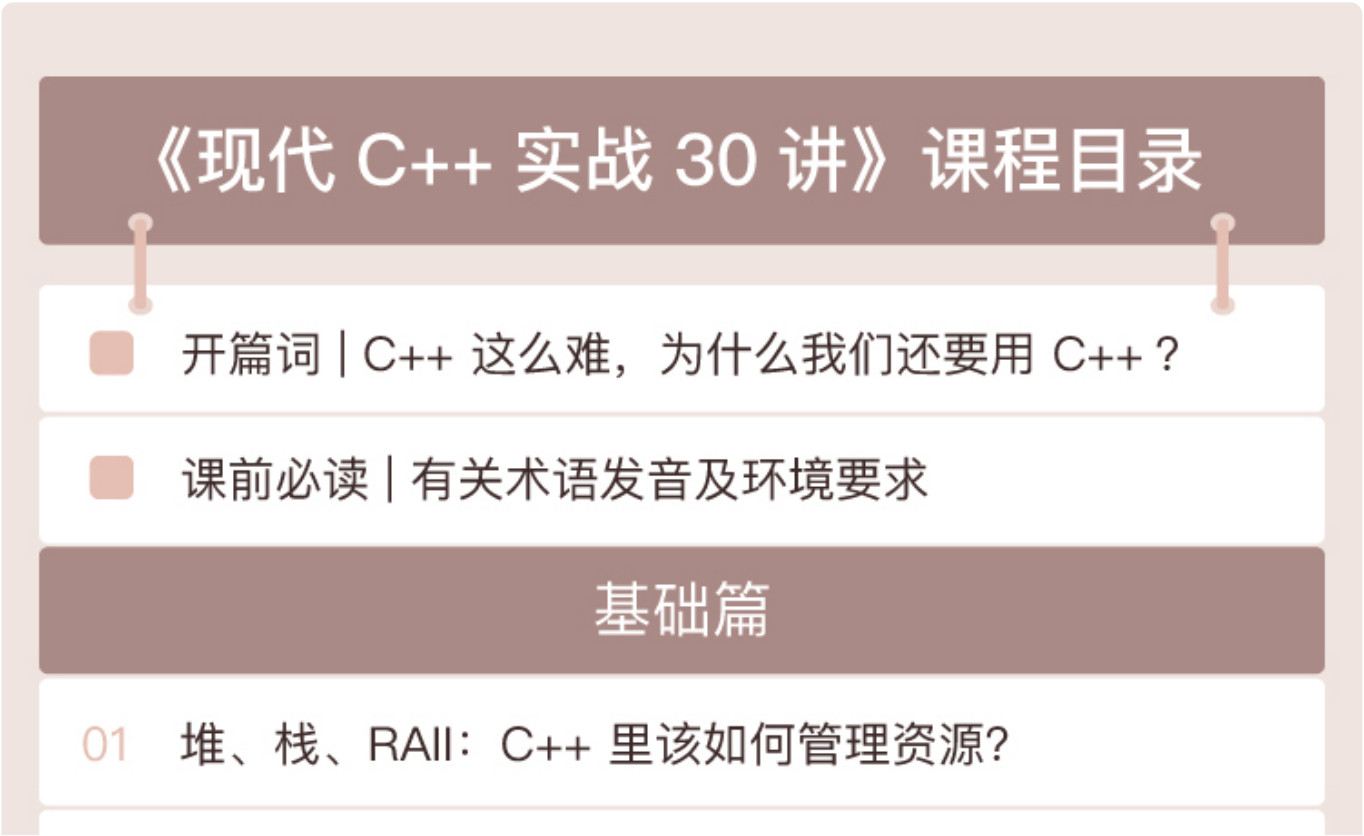
具体内容上，本专栏共分为四个部分，详情如下：

第一部分——基础篇，讲解现代 C++ 中的最重要特性，帮助你理解基础概念。

第二部分——提高篇，讲述几个独立的专题，帮助你掌握 C++ 中的一些高级技巧。

第三部分——实战篇，讨论实际的工具和第三方库，帮助你打磨手头的兵器库。

第四部分——未来篇，讨论 C++20 中即将引入的一些新特性，帮助你培养前瞻性。



02 自己动手，实现 C++ 的智能指针
03 右值和移动究竟解决了什么问题?
04 容器汇编 I：比较简单的若干容器
05 容器汇编 II：需要函数对象的容器
06 异常：用还是不用，这是个问题
07 迭代器和好用的新 for 循环
08 易用性改进 I：自动类型推断和初始化
09 易用性改进 II：字面量、静态断言和成员函数说明符

提高篇

10 到底应不应该返回对象?
11 Unicode：进入多文字支持的世界
12 编译期多态：泛型编程和模板入门
13 编译期能做些什么？一个完整的计算世界
14 SFINAE：不是错误的特化失败是怎么回事？
15 constexpr：一个常态的世界
16 函数对象和 lambda：进入函数式编程

17 函数式编程：一种越来越流行的编程范式

18 应用可变模板和 tuple 的编译期技巧

19 atomic、mutex 和 thread：进入并发的世界

20 future 和 promise：领略异步中的将来

实战篇

21 工具漫谈：编译、格式化、代码检查、排错各显身手

22 不同的数据和错误处理方式：optional、variant 和 expected

23 数字计算：介绍线性代数和数值计算库

24 Boost：你需要的“瑞士军刀”

25 两个单元测试库：C++ 里如何进行单元测试？

26 EasyLogging++：一个优秀的日志库

27 C++ REST SDK：使用现代 C++ 开发网络应用

未来篇

28 Concepts：如何对模板进行约束？

29 Ranges：无迭代器的迭代和更方便的组合

30 Coroutines：协作式的交叉调度执行

希望你在完整地学完这四个部分之后，能对现代 C++，这一熟悉而又陌生的语言，有着新的理解，并用它去更好地解决项目中的实际问题。

最后，感谢你的关注。关于 C++ 呢，不知道你有什么样的看法？你学习它的主要原因是什么？你平时使用 C++ 的场景有哪些？欢迎你在留言区和我交流。

参考资料

[1] 游戏葡萄，“《王者荣耀》技术总监复盘回炉历程：没跨过这三座大山，就是另一款 MOBA 霸占市场了”。http://youxiputao.com/articles/11842

[2] TIOBE Index for November 2019. https://www.tiobe.com/tiobe-index/

[3] Wenzel Jakob et al., pybind11. https://github.com/pybind/pybind11

[4] Interview, “What’s all the C Plus Fuss? Bjarne Stroustrup warns of dangerous future plans for his C++”. https://www.theregister.co.uk/2018/06/18/bjarne_stroustrup_c_plus_plus/

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

下一篇　课前必读 | 有关术语发音及环境要求

精选留言 (1)

写留言

黄振宇
2019-11-25

沙发，支持。好好跟着吴老师学习

展开